

# Characterizing the impact of last-level cache replacement policies on big-data workloads

Alexandre Valentin Jamet\*, Lluc Alvarez\*<sup>†</sup>, Daniel A. Jiménez<sup>‡</sup>, Marc Casas\*

\*Barcelona Supercomputing Center (BSC), <sup>†</sup>Universitat Politècnica de Catalunya (UPC), <sup>‡</sup>Texas A&M University  
 {alexandre.jamet, lluc.alvarez, marc.casas}@bsc.es, djimenez@acm.org

**Abstract**—The vast disparity between Last Level Cache (LLC) and memory latencies has motivated the need for efficient cache management policies. The computer architecture literature abounds with work on LLC replacement policy. Although these works greatly improve over the least-recently-used (LRU) policy, they tend to focus only on the SPEC CPU 2006 benchmark suite – and more recently on the SPEC CPU 2017 benchmark suite – for evaluation. However, these workloads are representative for only a subset of current High-Performance Computing (HPC) workloads.

In this paper we evaluate the behavior of a mix of graph processing, scientific and industrial workloads (GAP, XSBench and Qualcomm) along with the well-known SPEC CPU 2006 and SPEC CPU 2017 workloads on state-of-the-art LLC replacement policies such as Multiperspective Reuse Prediction (MPPPB), Glider, Hawkeye, SHiP, DRRIP and SRRIP. Our evaluation reveals that, even though current state-of-the-art LLC replacement policies provide a significant performance improvement over LRU for both SPEC CPU 2006 and SPEC CPU 2017 workloads, those policies are hardly able to capture the access patterns and yield sensible improvement on current HPC and big data workloads due to their highly complex behavior.

In addition, this paper introduces two new LLC replacement policies derived from MPPPB. The first proposed replacement policy, *Multi-Sampler Multiperspective (MS-MPPPB)*, uses multiple samplers instead of a single one and dynamically selects the best-behaving sampler to drive reuse distance predictions. The second replacement policy presented in this paper, *Multiperspective with Dynamic Features Selector (DS-MPPPB)*, selects the best behaving features among a set of 64 features to improve the accuracy of the predictions. On a large set of workloads that stress the LLC, MS-MPPPB achieves a geometric mean speed-up of 8.3% over LRU, while DS-MPPPB outperforms LRU by a geometric mean speedup of 8.0%. For big data and HPC workloads, the two proposed techniques present higher performance benefits than state-of-the-art approaches such as MPPPB, Glider and Hawkeye, which yield geometric mean speedups of 7.0%, 5.0% and 4.8% over LRU, respectively.

**Index Terms**—cache management, big data, graph processing, workload evaluation, micro-architecture

## I. INTRODUCTION

The vast disparity between main memory and CPU speed has led to hierarchical caching system in modern CPUs. The goal of the cache hierarchy is to keep data on-chip, close to the cores that are accessing it, thus avoiding hitting the memory wall [28]. Although computer architects highlighted the need for multiple levels to the cache hierarchy, the Last Level Cache (LLC) suffers from a high latency compared to the other cache levels. In addition, the LLC suffers from poor temporal and spatial locality in the access sequence as some accesses are

filtered by the upper levels of the hierarchy. This phenomenon is exacerbated when considering emerging workloads such as big data or graph-processing workloads displaying highly irregular behavior. Thus, emerging workloads require more sophisticated cache replacement policies that can cope with a broader set of workloads than the traditional ones.

State-of-the-art LLC replacement policies such as MPPPB [14], Glider [21], Hawkeye [10], SHiP [27], DRRIP, and SRRIP [12] show significant improvement when challenged by SPEC CPU 2006 [11] and SPEC CPU 2017 workloads. However, when facing workloads representative of another part of the spectrum of the HPC applications, these policies fail at delivering significant improvement over the baseline LRU policy. Such workloads with highly irregular behavior prevent the LLC replacement policies mentioned above from capturing the access patterns and, therefore, producing meaningful predictions and decisions. To address this issue, we argue that future work on LLC replacement policies should consider a more extensive set of workloads such as the one we study in this paper, which is composed of the following benchmark suites:

- the GAP benchmark suite [4].
- the XSBench benchmark suite [25].
- Qualcomm workloads for the CVPI [1] championship.

This paper also proposes two MPPPB variants that increase its benefits. First, we propose *Multi-Sampler Multiperspective (MS-MPPPB)*, a variant of MPPPB that uses four samplers and perceptron structures. MS-MPPPB adapts its replacement policy to the workload in a phase-wise manner, selecting the sampler that provides the best predictions out of the four available and using the most accurate sampler to make predictions and drive placement, promotion and bypass decisions in the LLC. Second, this paper proposes *Multiperspective with Dynamic Features Selector (DS-MPPPB)*, another variant of MPPPB that is also able to adapt its behavior to the execution phases of the workloads by dynamically selecting the most accurate subset of 16 features from a bigger pool of 64 features.

This paper makes the following contributions:

- 1) It evaluates state-of-the-art LLC replacement policies over a broader set of benchmark suites than usually considered in the literature. The selected benchmark suites better represent current and emerging big data and scientific workloads on HPC systems. This paper

considers the SPEC CPU 2006 and the SPEC CPU 2017 suites, a large set of workloads provided by Qualcomm for the CVP1 championship, the GAP benchmark suite, and the XSBench. This paper also takes the opportunity to build knowledge on these workloads and analyzes their behavior and impact on the LLC and the memory hierarchy, thus paving the way for further work.

- 2) We present MS-MPPPB and DS-MPPPB, two novel LLC replacement policies derived from MPPPB. The main idea behind both schemes is to improve the accuracy of the predictions by dynamically selecting the most accurate features for each phase of the running workload. On a set of 50 cache intensive benchmarks, these new designs respectively yield a geometric mean speed-up of 8.3% and 8.0% over LRU, and outperform all the state-of-the-art approaches.

The rest of this paper is organized as follows: Sections II and III describe the workloads and the state-of-the-art replacement policies evaluated in this paper, respectively. Section IV motivates the need for additional benchmarks in the evaluation of LLC replacement policies. Section V proposes MS-MPPPB and DS-MPPPB, two designs derived from MPPPB that achieve higher accuracy on the studied benchmarks. Section VI defines the evaluation methodology. Section VII presents the results of our experiments and comment on them. Finally, Section VIII remarks the main conclusions of this work.

## II. WORKLOADS

Benchmarks are of paramount importance for the computer architecture community, as they are used in practically all the stages of processor development, from the very initial research to the final performance verification of the processors that are manufactured for all market segments, from embedded devices to the most powerful supercomputers in the world. Benchmark suites are composed of a series of codes and representative input sets, and their goal is to mimic the behavior of real workloads to define the performance goals of a processor design and to bring to light unexpected design issues. Hence, their choice is of crucial importance. This section presents a set of benchmarks that are commonly used by the community to model the behavior of different types of workloads.

### A. SPEC CPU Benchmark Suites

The SPEC CPU benchmark suite [11] is a set of benchmarks aimed at studying the performance of CPU designs. These benchmarks are well-known and highly used by the computer architecture community, specially to evaluate new proposals in the area of microarchitecture. These benchmarks provide representative codes of real compute intensive workloads such as compilers, data compression, AI algorithms, and physics. These workloads are mostly scientific applications or commonly used algorithms in computer sciences such as data compression and parsers which loop over data structures in a reasonably predictable manner, which allows the cache structures of the CPUs to leverage the locality of these workloads. However, although these benchmarks cover a broad spectrum

of applications, they do not represent some codes running on current HPC systems and mobile devices.

### B. GAP Benchmark Suite

To help to standardize the evaluation of big data and graph processing algorithms, Beamer *et al.* proposed the GAP Benchmark Suite [4], a set of domain specific workloads that include graph computational kernels as well as representative input graphs. These domain specific workloads provide computer architects with the ability to extend their working sets of workloads. The benchmark suite provides a standardized evaluation framework for commonly used graph algorithms such as *Page Rank* and *Connected Components*, along with standard graph inputs available in industry and research.

1) *Graph kernels*: Next we provide a short description of each of the six graph kernels available in the benchmark suite.

*Breadth-First Search (BFS)* was proposed in 1945 by Konrad Zuse and it is one of the most well-known and widely used graph processing algorithms. Its principle is rather simple, and it comes down to a straightforward statement: first, one designates a root vertex to initiate the search algorithm, then the kernel traverses all the neighbouring vertices before moving to the next depth level.

*Single Source Shortest Path (SSSP)* is a prevalent problem in graph theory and engineering in general. This algorithm computes the distance to any reachable vertices from a given source vertex, being the distance between two vertices the minimum sum of edge weights along a path connecting the two vertices.

*Page Rank (PR)*, invented by Larry Page to quantify the popularity of a web page, is a widespread algorithm in our daily life as it allows search engines to build meaningful proposals to our questions. It is an iterative algorithm that associates a score (a PageRank) to each vertex of the graph. During an iteration, the algorithm updates the score of every vertex proportionally to the sum of the scores of its incoming neighbourhood. The algorithm stops when the variation of PageRanks in the graph falls below a limit, which means that the sum of the variations of the scores of all the vertices between two steps is below a certain threshold.

*Connected Components (CC)* is an algorithm meant to identify and label connected components in a graph. A connected component is a sub-graph in which its paths connect any two vertices, and the vertices of the sub-graph are not connected to any other vertex in the super-graph.

*Betweenness Centrality (BC)* is a crucial concept in graph and network theory that allows measuring the influence of a vertex in the data transfers of a network, assuming ideal transfers through the shortest paths.

*Triangle Count (TC)* is an algorithm that is mostly used in social network analysis to detect communities by detecting triangles in a graph. Triangles are a group of three vertices directly connected.

2) *Input graphs*: The GAP Benchmark suite comes with five inputs graphs of diverse origin (synthetic versus real world). The real world data sets model the connection between

people, websites and roads. When selecting these real world graphs, the authors paid particular attention to the size of the graphs so that they can fit in the memory of most servers while stressing the cache hierarchy of such systems.

*Twitter* is a crawl of Twitter that has been commonly used by researchers to evaluate prior work and thus allows fair comparisons. It allows working with a typical example of social network topology, and its real world origin gives it interesting properties such as irregularities.

*Web* is a web crawl of the .sk domain. Even though it has a large size, it exhibits good locality and high average degree.

*Road* is an input graph that models the distances of all of the roads in the USA. Although it has a modest size compared to the other graphs available, it has a rather high diameter that can cause some algorithms to present large execution times.

*Kron* provides continuity with prior work as it has been used frequently in research. This graph uses the Kronecker synthetic graph generator.

*Urund* represents a worst case scenario as all vertices have an equal probability of being a neighbour of every other vertex.

### C. XSBench workloads

XSBench workloads [25], as stated by their authors Tramm and Siegel, are meant to represent the *most computationally intensive steps of a robust nuclear core Monte Carlo particle transport simulation*. These workloads provide a variety of grid types, sizes and browsing algorithms, which allows computer architects to stress the memory hierarchy of a CPU in different ways and to expand their working set of workloads towards new scientific applications.

The XSBench suite allows customizing the code that will be effectively executed in order to stress the memory hierarchy. The benchmark suite relies on a handful of parameters to achieve this flexibility. In this work we focus on the three parameters that put more pressure on the cache hierarchy and, thus, have a higher significance for this work.

1) *Problem size*: When solving the particle transport problem, the size of the problem has a dramatic influence on performance and on the stress that is being put on the memory hierarchy. Eventually, increasing the size of the problem has a significant impact on performance as data structures are much larger, so we use the two largest sizes of grid available.

2) *Grid type*: This parameter allows the user to select among three types of grids. The *nuclide* grid is known as a naive implementation and does not require any additional memory other than what is necessary to store the point-wise cross-section data. However, it is computationally intensive as the benchmarks execute a binary search with high frequency. *Unionized* is a grid type that allows for higher performance as it uses an acceleration structure to reduce the number of binary searches triggered during the execution. Here, this optimization sacrifices memory footprint to leverage increased performance. The *hash* grid is presented as a competitive alternative to the unionized grid type as it allows to achieve similar performance while using far less memory.

3) *Number of cross-section look-ups*: This parameter sets the number of look-ups to perform per particle.

### D. Industrial workloads

During the CVP1 contest, the evaluation of Value Prediction mechanisms used a set of over 2000 industrial workloads provided by Qualcomm. These are typical server and database workloads such as Redis and MongoDB, among others. Real world database workloads traverse vast amounts of data while processing a query and show low reuse of data over time. Thus, these workloads are known to be memory intensive and they stress the LLC more than the SPEC CPU 2006 and the SPEC CPU 2017 workloads.

## III. CACHE REPLACEMENT POLICIES

While developing new cache replacement algorithms for LLCs, one needs to evaluate the policy against a set of workloads that show the behavior of interest. This section reviews the most relevant cache replacement algorithms designed for LLCs. As this work studies the impact of emerging workloads on the LLC, we present the state-of-the-art replacement policies developed for this specific cache level. The cache replacement problem is slightly more complex in the context of the LLC than in the context of L1 and L2 caches. Although the underlying idea remains the same, the LLC suffers from poor locality as the upper-level caches filter accesses and leave only a cluttered sequence to the LLC. To cope with this particular replacement problem, researchers have come up with more and more sophisticated design ideas to leverage higher prediction accuracy and performance.

The next subsections present the most relevant state-of-the-art work on LLC replacement policies.

### A. Reuse Distance Prediction

As reuse distance is a crucial concept when it comes to cache replacement, recent works focus on proposing new techniques to build run-time approximations of the distance to the next reuse of a cache block. Re-reference Interval Prediction (RRIP) and all its derivatives are efficient yet lightweight implementations of reuse distance prediction.

The main idea behind RRIP is the classification of blocks into re-reference classes. In their work, Jaleel *et al.* propose three versions of the RRIP replacement policies [12], [20]: SRRIP, BRRIP, and DRRIP. The former, scan-resistant, is limited to always inserting new coming blocks in a fixed class. In contrast, BRRIP provides more flexibility by frequently inserting blocks in the distance re-reference class and infrequently inserting blocks in the long re-reference class. Finally, DRRIP leverages Set-Dueling to determine which of SRRIP and BRRIP is best suited for a given workload or program phase, making it both scan and thrash resistant.

### B. Signature-based Hit Predictor

Building on the reuse distance prediction [12], [20] framework built by Jaleel *et al.* and program-counter based dead

block prediction [16], Wu *et al.* [27] proposed a LLC replacement policy design that uses a program-counter based signature as a feature.

As stated while describing reuse distance prediction mechanism in Section III-A, SRRIP learns the re-reference intervals of the living cache blocks relatively to one another. The primary feature of *Signature-based Hit Predictor (SHiP)* [27] is that, not only it allows the SRRIP policy to learn the relative re-reference intervals, but it also tries to learn the likelihood of cache blocks to experience hits through a feature. The intuition being that cache blocks with the same signature behave comparably. In order to learn the likelihood of a cache block to experience further hits, SHiP maintains a prediction table with an entry per signature. When a signature gets hit, the associated saturating counter is incremented. Conversely, when a signature misses, the associated counter is decremented.

With the prediction values thus learned, SHiP modifies SR-RIP policy for insertion by inserting new coming cache blocks in the distant re-reference interval if the prediction associated with the signature of that cache blocks is zero. A zero in the prediction table gives a strong hint that the associated signature belongs to the distance re-reference interval.

### C. Multiperspective Reuse Prediction

The *Multiperspective Reuse Prediction* [14] cache replacement algorithm (hereafter MPPPB) leverages perceptron learning for reuse prediction and drives placement, promotion and bypass decisions. This replacement policy extends the idea of features developed in previous work [16], [24], [27] to achieve higher accuracy. It is essentially made of two components, a sampler and a perceptron predictor. The sampler, based on observations of block evictions relatively to its features associativity, is responsible for triggering learning signals to the perceptron. The perceptron, based on the learning signals triggered by the sampler, updates its prediction tables.

MPPPB relies on the idea of correlating reuse prediction of a cache block with a large number of features that ranges from PCs to characterizing bursty access patterns. In this context, a feature can be defined as a hash function applied to cache block characteristics such as the PC or the physical address. When a prediction request occurs, the perceptron selects weights out of its prediction tables using hashes of multiple features. Each feature is hashed to index its prediction table, and the weights obtained are gathered in a single prediction value by a simple addition and compared to a set of thresholds to drive actions such as bypass, promotion and placement.

Perceptron learning is used to update the weights of the prediction tables through the learning algorithm. At the time a sampled block is reused or evicted, the perceptron updates the weights of the prediction tables associated with the last access to this block, according to the perceptron learning rule. For instance, if a block hits in the sampler while having its LRU stack position lower than the associativity of a feature, it is trained positively for that feature. Conversely, if a block gets demoted beyond the associativity of a given feature, it is trained negatively for that feature.

With this work, Jiménez and Teran demonstrated the usefulness and impact of combining multiple features. Among the correlating features, the sequence of PCs leading to the usage of a block is one; however, the sequence of PCs is highly filtered by the other levels of the cache hierarchy, making it inaccurate for predictions. The introduced additional features such as bits extracted from the memory address help mitigating the inaccuracy of a filtered PC sequence. MPPPB relies on this idea of combining multiple features while significantly augmenting the set of available features.

### D. Optimal Replacement Approximation

The Hawkeye [10] replacement policy marked the birth of a new class of cache replacement algorithms aiming at approximating, in a relatively affordable way, optimal but unimplementable algorithms such MIN [6].

Hawkeye and its successor, Glider [21], are primarily made of two major building blocks: an optimal solution approximation component and a predictor that learns from the former component. The predictor is used to compute predictions and to trigger actions based on these predictions. The first component provides a binary output about the cache block of interest: needs to be cached or not. For this outcome, the predictor gets trained for the associated PC as it is a PC-based predictor. When the replacement policy requests a prediction to drive its decision making, the predictor is indexed, and it uses its outcome to place blocks in the matching RRIP categories, thus prioritizing eviction for blocks classified as cache-averse. Conversely, blocks identified as cache-friendly tend to stay in the immediate-reuse category.

Further work on the Hawkeye predictor provided it with a more complex predictor infrastructure. That infrastructure, named Glider, leveraged on the knowledge obtained through the offline training of a machine learning model, yielding additional performance improvements.

## IV. MOTIVATION

To highlight the need for new benchmarks in the context of the development of new cache replacement policies for LLCs, we provide a quantitative analysis to build intuition on why the current state-of-the-art techniques need to take into account a broader set of workloads in the process of their constructions. This analysis relies on results obtained using the simulation methodology detailed in VI.

Figure 1a shows the average LLC MPKI for each of the benchmark suites described in Section II using the baseline LRU replacement policy. In both Figures 1a and 1b we only consider the cache-intensive benchmarks of these benchmark suites, namely the ones which present a LLC MPKI over 1.0 with the baseline LRU replacement policy. The GAP benchmark suite and all the different runs of XSBench, with respective LLC MPKI of 78.29 and 36.62, provide a significantly higher LLC MPKI than what is provided by the SPEC CPU benchmarks. The industrial Qualcomm workloads and the SPEC CPU workloads, with respective LLC MPKI of 10.63 and 15.76, do not show such a high impact on the LLC.

These results demonstrate that big data graph processing workloads like the ones modeled in the GAP benchmark suite highly stress the cache hierarchy, and particularly the LLC, much more than well-known workloads such as SPEC CPU 2006 and SPEC CPU 2017 suites do. This is due to the nature of these workloads, where moving from edge to edge in a graph structure leads to extremely unpredictable and sparse access patterns [29], [30]. Also, the memory footprint of the inputs is an important factor, as jumping from edges to edges in an extensively large graph exhibits very low spatial and temporal locality, which are two key concepts in the design of cache replacement policies.

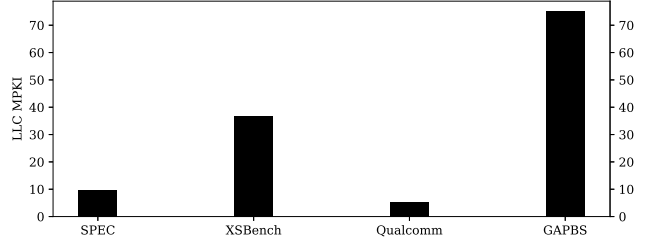
Figure 1b shows the speed-up of the state-of-the-art cache replacement policies presented in Section III over the baseline LRU policy for the different benchmark suites presented in Section II. Each bar represents a single replacement policy, and each bar group stands for a benchmark suite. Results show that the different policies can catch a different kind of access patterns and are beneficial for different kind of workloads.

For the SPEC benchmarks, the plot shows that every single replacement policy is consistently delivering improvements over the baseline LRU and the previously proposed replacement policies in the literature, incrementally improving the performance of these benchmarks.

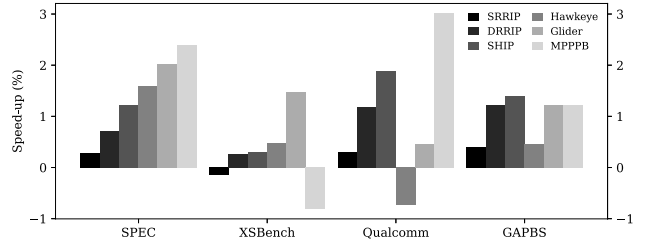
The replacement policies based on Reuse Distance Prediction (SRRIP and DRRIP) are consistent in their improvement over the baseline LRU, while more complex policies such as SHiP, Hawkeye, Glider and MPPPB have more difficulties generalizing to all the benchmark suites. The main reason behind this observation is that all these replacement policies rely either on assumptions about the access patterns (*e.g.*, SHiP and Hawkeye) or on a training phase over a set of workloads (*e.g.*, Glider and MPPPB). On the one hand, SHiP and Hawkeye use the observation that they can accurately learn the access patterns to the LLC using the PCs that triggered the memory accesses as a classification feature. On the other hand, both Glider and MPPPB rely on a learning algorithm that learns the access patterns of a couple of workloads and provide correlating features such as the  $i$ -th PC of the history or some bits of the physical address of the accessed block. Thus, although these state-of-the-art replacement policies deliver some performance improvements, they suffer from a structural bias that prevents them from generalizing to unexplored benchmarks in an optimal way.

The main conclusions arising from our analysis are:

- 1) The commonly used SPEC CPU 2006 and SPEC CPU 2017 suites no more represent a challenge for computer architects, as they are well studied and there are plenty of ingenious mechanisms that cope with their behavior.
- 2) The current state-of-the-art LLC replacement policies do not generalize well to new benchmarks.
- 3) Emerging big data and HPC workloads do represent a challenge for computer architects, as they stress more the cache hierarchy than traditional workloads. Nevertheless, they reveal the need to take into account their behavior in the design of forthcoming CPUs.



(a) Average LLC MPKI of the different benchmark suites using the LRU replacement policy.



(b) Geometric mean speed-up over LRU of state-of-the-art LLC replacement policies for the different benchmark suites.

Fig. 1. LLC MPKI using the LRU replacement policy and performance improvement of state-of-the-art LLC replacement policies for the different benchmark suites.

## V. DESIGN PROPOSALS

Along with the state-of-the-art cache replacement policies presented in Section III, we introduce MS-MPPPB and DS-MPPPB, two new LLC replacement policies derived from the original MPPPB. These two policies try, in distinct manners, to adapt themselves to the behavior of the emerging big data and HPC workloads.

### A. Multi-Sampler Multiperspective

With MPPPB, Jiménez and Teran provide a replacement policy based on a reuse predictor, which ultimately relies on a hashed perceptron table.

Our first proposed design, named MS-MPPPB, is based on the idea that having not just one but many hashed perceptron tables can yield higher prediction accuracy and improved performance by dynamically choosing one of the perceptrons to trigger predictions. To perform the selection of the best behaving perceptron that will eventually trigger predictions, all perceptrons are competing against each other through a two-rounds decision tree scheme [15] that duels each of the four available perceptrons and selects the one that minimizes misses in the LLC. Although not used to produce a prediction, the three perceptrons left unused are concurrently updated following the process described by Jiménez and Teran in *Multiperspective reuse prediction* [14].

The additional hardware budget required for this proposal is rather high, as a naive implementation would lead to the instantiation of 4 individual samplers along with the 4 perceptron tables bound to them. Each block of the sampler holds an indices trace of the last accessed elements of the prediction tables, which requires a maximum of 128 bits. For each block, the sampler holds a 16-bit partial tag along with

a 5-bit LRU state and a 9-bits confidence value. The sampler takes the form of a cache with 80 sets and 18 ways.

### B. Multiperspective with Dynamic Features Selector

In the second design proposal, named DS-MPPPB, we use an additional concept along with the already existing idea of weights. Our new concept, *Coefficients*, revisits the conception of a hashed perceptron [22] by introducing the weights used in the mathematical definition of a perceptron [26].

We thus differentiate two key concepts. The *weights* are the actual values contained in the prediction tables of a hashed perceptron. These values are meant to reflect the learned reuse distance based on the observation of past events. These events can be the occurrence of a specific PC, a physical address or any other source of information used as feature [3], [16], [18]. The *coefficients* are confidence counters that reflect how accurate is the prediction table bound to a specific confidence counter.

The original code of MPPPB is shipped with not just one set of features but four, which adds up to a total of 64 features. Each of these sets of features was developed following the hill-climbing methodology described in *Multiperspective Reuse Prediction* [14] and is designed to fit each of the possible configurations of the CRC2 contest.

We gather all the features in a single set and build a perceptron predictor using them all. Although we now have a set of 64 features, we want to select only the 16 best behaving ones among the 64 available. To do so, for each prediction triggered by the replacement policy, the predictor searches for the 16 features with the highest confidence values and uses them to build a prediction, and the other features are left unused for that prediction. However, confidence values of all features are updated following algorithm 1.

---

#### Algorithm 1 Updating confidence values of prediction tables

---

```

hit ← false
truth ← 0 $\mathbb{B}^n$  {A n-vector of falses.}
pred ∈  $\llbracket -32; 31 \rrbracket^n$  {A vector of individual predictions.}
if Accessed block hits in the sampler then
    hit ← false
else
    hit ← true
end if
for all  $i$  such that  $0 \leq i \leq n - 1$  do
    truth[ $i$ ] ← ((pred[ $i$ ] < 0) = hit)
    if truth[ $i$ ] = true then
        conf $i$  ← max(conf $i$  + 1, confmax, $i$ )
    else
        conf $i$  ← min(conf $i$  - 1, 0)
    end if
end for

```

---

For clarity, we include a summary of the notations we use.  $\mathbb{F}$  denotes the set of features,  $n$  is the total number of features and  $m$  the number of features we include in the prediction value. We denote the confidence counter of the  $i$ -th feature

as  $conf(f_i) = conf_i$  along with the upper bound of the confidence counters  $conf_{max,i}$ . We denote as  $t_i$  the prediction table associated with feature  $f_i$ .

We thus define  $\check{\mathbb{F}}$ , the set of all possible arrangements of unique  $m$  features taken out of  $\mathbb{F}$  and  $\check{\mathbb{F}}_{max}$  the element of  $\check{\mathbb{F}}$  that maximizes the sum of confidence counters. Finally, we compute the prediction value by summing the weights taken out of the tables of the elements of  $\check{\mathbb{F}}_{max}$ .

Table I summarizes the hardware budget of each design proposal described in this section. Along with the total hardware budget required for each proposal, we also provide the budget required by each component, namely: the replacement states (here we use MDPP, a modified Tree-based PLRU [13], [23] policy that uses a custom transition vector to determine to which position an accessed block should be moved to), the sampler(s) and perceptron(s).

	Replacement states	Sampler(s)	Perceptron(s)	Total
MS-MPPPB	3.75 KiB	111.09 KiB	12 KiB	126.84 KiB
DS-MPPPB	3.75 KiB	95.27 KiB	12 KiB	111.02 KiB

TABLE I  
HARDWARE BUDGET OF THE PROPOSED DESIGNS.

## VI. METHODOLOGY

In this section we present the evaluation methodology used to report results in Section VII. In particular, the next subsections present the set of workloads used to evaluate the different LLC replacement policies and our workload selection methodology, a description of the simulation environment, and the evaluated replacement policies and their configuration.

Overall, we follow the same evaluation methodology as the one used by Shi *et al.* [21] with the aim of building the fairest comparison possible against state-of-the-art techniques.

### A. Workloads

For the evaluation of the different LLC replacement policies we consider the following sets of workloads:

- Over 2000 Qualcomm workloads used for CVP1 contest.
- All SPEC CPU 2006 and CPU 2017 benchmarks.
- All workloads included in the GAP Benchmark Suite.
- All workloads included in the XSBench Suite.

From all these benchmarks we select the 50 most intensive workloads so that our evaluation set of workloads is a blend of each suite designated above. We use the SimPoints [19] methodology to identify intervals (hereafter *SimPoints*) representative of at least 5% of the SPEC, GAP and XSBench workloads. Each SimPoint is 1 billion instructions long and characterizes a different phase of these workloads. Each SimPoint is executed for 200 million instructions in order to warm-up the memory hierarchy, and then it is executed for an additional 1 billion instructions to report experimental results.

We only evaluate these workloads in a single-thread context. We deliberately chose to restrict our evaluation to single-core as this work focuses on the characterization of the access patterns of the selected workloads to the LLC. The modeled architecture being composed of a shared LLC, modeling an architecture using multiple cores we would not be able to

Component	Description
Branch Predictor	hashed perceptron
CPU	4 GHz, 4-wide out-of-order processor 6-stage pipeline, 128-entries re-order buffer
L1 ITLB	64-entry, 4-way, 1-cycle latency, 8-entry MSHR
L1 DTLB	64-entry, 4-way, 1-cycle latency, 8-entry MSHR
L2 TLB	1536-entry, 12-way, 8-cycle latency, 16-entry MSHR
L1-I Cache	32 KiB, 8-way, 4-cycle latency, 8-entry MSHR
L1-D Cache	32 KiB, 8-way, 4-cycle latency, 8-entry MSHR next line prefetcher
L2 Cache	256 KiB, 8-way, 12-cycle latency, 16-entry MSHR ip-stride prefetcher
LLC	2 MiB, 16-way, 26-cycle latency, 32-entry MSHR
DRAM	4 GiB, DDR4 SDRAM data-rate: 3.2 GT/s, I/O bus frequency: 1.6 GHz $t_{RP} = t_{RCD} = t_{CAS} = 24$ cycles

TABLE II  
SYSTEM SIMULATION PARAMETERS.

properly measure reuse distances as the different cores would be asking for distinct data in the same cache, thus compromising our measurements. The results reported per benchmark (for SPEC, GAP and XSBench) are the normalized weighted averages of the results for individual SimPoints. In contrast, the Qualcomm workloads are single-trace benchmarks that do not use such methodology.

### B. Experimental setup

Our evaluation considers ChampSim [7], a detailed trace-based simulator that models an out-of-order CPU along with its cache hierarchy, prefetching mechanisms and memory subsystem. Table II provides a detailed configuration of the modeled CPU and the memory hierarchy.

### C. Replacement policies simulated

We evaluate the workloads described in Section II against the most relevant cache replacement policies proposed in the literature: SRRIP, DRRIP, SHiP, MPPPB, Hawkeye and Glider, all detailed in Section III. Although there is a vast amount of work in reuse prediction available in the literature [2], [5], [8], [12], [13], [16], [17], [23], [24], [31], the aforementioned replacement policies that have been selected for the evaluation are the most recent and relevant approaches in the state-of-the-art. In addition, in the evaluation we also include the two new replacement policies proposed in this paper, explained in Section V. These two new techniques are derived from MPPPB and leverage the usage of multiple perceptrons to achieve higher accuracy. For MPPPB we used the code that is publicly available on the website of the CRC2 contest [9]. For Glider we use code graciously provided by the authors.

## VII. RESULTS, ANALYSIS AND DISCUSSION

This section presents our experimental campaign along with the results obtained and the characterization of the studied workloads. Section VII-B presents the performance benefits yielded by the different state of the art cache replacement policies mentioned in Sections VI-C and V. Section VII-A studies the impact of these workloads on the LLC in terms of misses and presents the MPKI reduction obtained by the

replacement policies mentioned above. Finally, Section VII-C studies the behavior of the studied workloads via the reuse distance of the cache blocks to highlight the different behavior of these benchmark suites.

### A. Misses

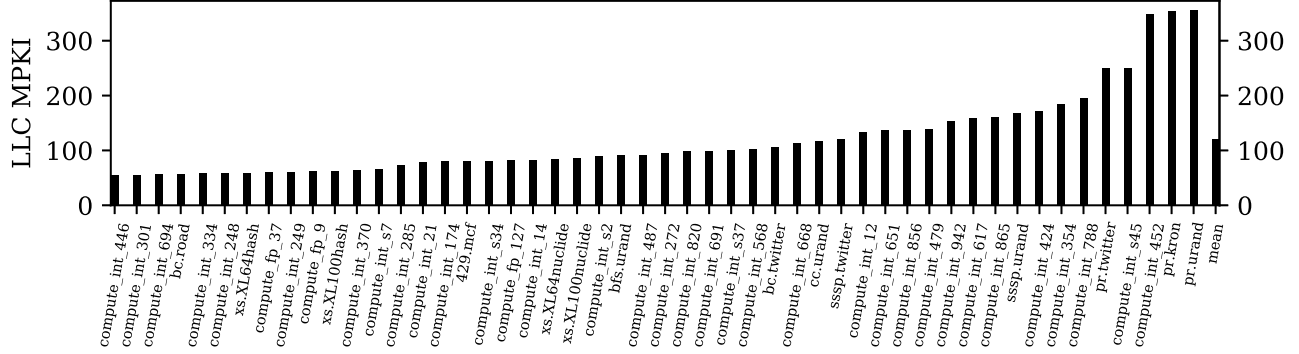
Figure 2a shows the LLC MPKI of the 50 most intensive workloads using the LRU replacement policy. These workloads show a very high impact on the LLC, with an average MPKI of 120, much larger to the workloads used in previous work. We use this to define the set of workloads that will be used in Section VII-B to evaluate state-of-the-art replacement policies along with our custom designs. These results clearly show the high impact on the LLC of Qualcomm, GAP and XSBench workloads. This selection of benchmarks is comprised of each of the studied benchmark suites and allows to evaluate replacement policies against a broad range of workloads with different behaviors.

### B. Performance

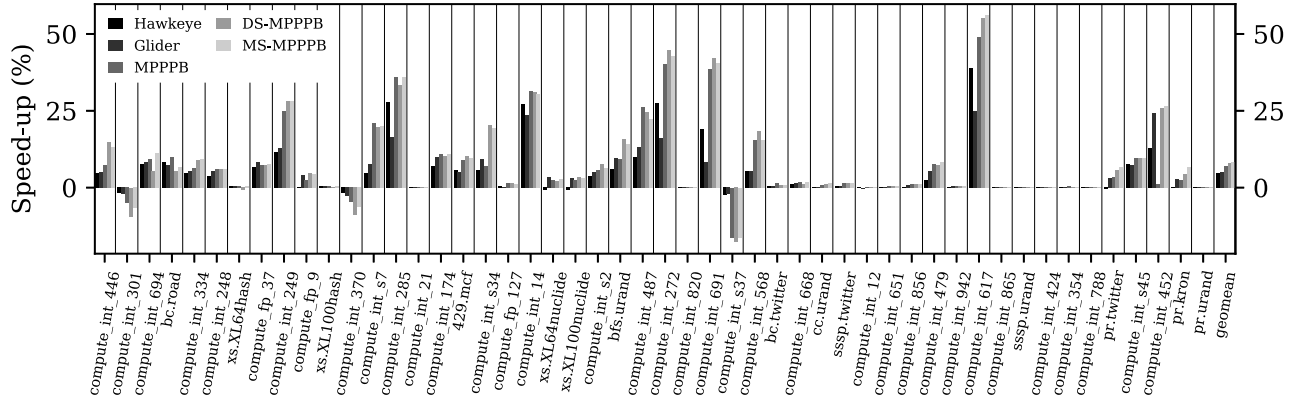
Figure 2b shows the speed-up of various replacement policies presented in Section VI-C and in Section V. LLC MPKI sorts the benchmarks with a baseline LRU policy. While Figure 1b was showing Glider standing out against MPPPB in some situations, Figure 2b clearly shows the versatility of the former and its ability to consistently deliver good performance even facing the most intensive and hard to predict workloads. MPPPB and Glider provide respectively 7.0% and 5.0% speed-up over baseline LRU. Thus, Glider is not able to deliver significant improvement over the baseline LRU compared with MPPPB. This fact suggests that the machine learning algorithm used to design the predictor of Glider should be trained against a wider variety of workloads, hence the need to include the workloads used in this paper in further work on LLC replacement policies.

Besides, we justify the difference of performance improvement in Figures 1b and 2 between Glider and MPPPB compared to the results published in *Applying Deep Learning to the Cache Replacement Problem* [21] by a difference in the methodology. As a matter of fact, we use the SimPoint methodology to generate at most ten simpoints and we only use the ones accounting for more than 5% of the whole execution, whereas the original results of Glider were reported with only a single trace per benchmark. We argue that our methodology is more robust as we cover more distinct behaviors, thus challenging the different techniques studied.

We also report significant performance improvements for our proposed techniques derived from MPPPB. The two designs proposed in this paper, MS-MPPPB and DS-MPPPB, respectively yield 8.3% and 8.0% speed-up over the baseline LRU, and both new approaches also outperform MPPPB. We observe that this performance improvement is due to the ability to adapt the set of features used for prediction in a execution phase-wise manner, providing more versatility to the design than the previously discussed techniques.



(a) LLC MPKI using the LRU replacement policy for the 50 most intensive workloads.



(b) Speed-up over LRU of the state-of-the-art LLC replacement policies and our custom replacement policies for the 50 most intensive workloads.

Fig. 2. LLC MPKI using the LRU replacement policy and performance improvement of LLC replacement policies for the 50 most intensive workloads.

### C. Reuse distances

Figure 3 shows the distribution of reuse distances for each benchmark suite used in this work. Figure 3 is organized as a box plot; the horizontal line of each distribution representing its median, the box ranging from the first to the third quartiles and the whiskers representing extreme values through the first and ninth deciles of the distributions. For readability purposes, we cut the box plot to only show whiskers but not flyers. For instance, the distribution of reuse distances of the GAP workloads have ninth decile around 2000 and a median of 116. We complete figure 3 with Table III as it provides additional characteristics of the distribution such as maximum, median, mean and standard deviation of the observed reuse distances.

In this work, we define reuse distance as the number of accesses to different cache blocks between two consecutive accesses coming from the CPU (*e.g.* read and writes but not prefetches and write-backs) to the same cache block. Looking at this figure, it is quite clear that all the benchmark suites suffer from the presence of dead blocks in their access patterns. However, having a closer look at each of the distribution presented can provide us with valuable knowledge and give intuition about the results obtained in Sections VII-B and VII-A. Blocks that do not experience a single reuse during the whole execution of a workload are not represented on Figure 3. However, in order to still provide that valuable information,

we show in Table III the average proportion of accesses to dead blocks during the execution.

For the rest of this section, we define as *cache-friendly blocks* those blocks with a reuse distance lower than the associativity of the LLC. Conversely, we define *cache-averse blocks* as blocks with a reuse distance higher than the LLC associativity. Assuming an LRU policy for the cache, if a cache block A gets inserted in the LLC and the cache sees 16 accesses to different blocks before re-referencing block A, the second access to block A will cause a miss because it will have been evicted. This block A occupies part of the cache capacity that can be used to allocate another block, hopefully cache-friendly, and can provide a hit instead of miss.

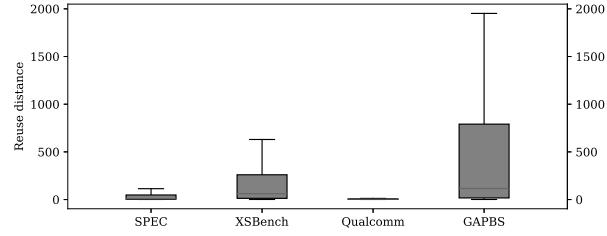
1) *SPEC benchmarks*: As Figure 3a and Table III show, the SPEC workloads present an average reuse distance of 126 and a standard deviation as low as 572.94, so the aggregated distribution of reuse distances for these workloads focuses mainly on low values with a relatively low standard deviation which translates into a cache-friendly behavior. Also, most accesses being cache-friendly, a bypass policy is not required to achieve reasonable performance over LRU.

This behavior explains why policies such as Hawkeye and Glider perform well when applied to SPEC workloads. These policies are focusing on prioritizing eviction for blocks that previously showed cache-averse behavior. Doing so allows to free space for cache-friendly blocks, however, as we will see

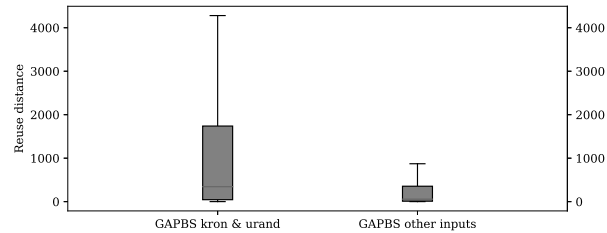


Parameter (in accesses)	SPEC	XSbench all	XSbench <i>unionized</i>	XSbench others	Qualcomm	GAP all	GAP <i>kron &amp; urand</i>	GAP others
Maximum	21 076	12 677	2248	12 677	59 934	34 836	33 771	34 836
Median	10	61	84	55	4	116	45	27
Mean	126.23	343.23	193.21	385.36	93.76	907.89	880.88	349.25
Standard deviation	572.94	875.68	256.33	977.54	541.80	2106.41	1885.68	1050.23
Avg. proportion of dead block accesses	48.03 %	37.32 %	46.63 %	28.01 %	10.54 %	55.84 %	59.31 %	54.06 %

TABLE III  
MAIN PARAMETERS OF THE DISTRIBUTION OF REUSE DISTANCES FOR CACHE BLOCKS IN THE DIFFERENT BENCHMARKS.



(a) Distribution of the reuse distances of the cache blocks for the different benchmark suites.



(b) Distribution of the reuse distances of the cache blocks for the GAP benchmark suite with different input sets.

Fig. 3. Distribution of the reuse distances of the cache blocks for the different benchmark suites.

in details for GAP benchmarks and XSbench benchmarks in the next subsections, replacement policies in general are less effective when the memory footprint increases and the access patterns become unpredictable.

2) *XSbench benchmarks*: The *XSbench* benchmark suite, as expected based on the reuse distance characteristics established in Table III, shows a very distinct behavior from the well-known *SPEC* CPU 2006 and *SPEC* CPU 2017 workloads. As a matter of fact, the *XSbench* workloads present an average reuse distance of 343.23 and a standard deviation as high as 875.68, which reveals a hard to predict behavior. However, they do experience less dead accesses to the LLC, the proportion of dead block accesses being 37.32 %.

With both mean and median reuse distances being the double of the *SPEC* ones, it is clear that these workloads are much more biased towards a cache-averse behavior than the *SPEC* workloads. However, with such high distribution parameters, we can still make a crucial observation when it comes to reuse prediction and dead blocks. We observe that, even though the *XSbench* workloads experience reasonable reuse of cache blocks that are accessed more than once during the execution, this happens for all the grid types except the *unionized*, which has a substantially larger memory footprint than the other grid types. We explain such behavior by the vast amount of data that the solver needs to traverse during the workloads execution and the algorithms used to do so.

3) *Qualcomm benchmarks*: When it comes to Qualcomm workloads, we observe that the distribution of reuse distances stretches towards higher values (Table III shows that the tail of the distribution goes as high as 60 000 accesses). However, these workloads are relatively biased towards a cache-friendly behavior as the standard deviation is 541.80 and the dead block accesses proportion is of 10.54 %. This stretched shape leads to a rather low standard deviation and to the appearance of low reuse distances with higher probability than of high reuses distances. We understand the long-wide tail of the distribution as a low amount of dead blocks thrashing the LLC, using space that would be a better fit for cache-friendly blocks.

Thus, this observation gives us intuition about the behavior of Glider on the Qualcomm benchmarks presented in Figures 2 and 1b. Glider is a replacement policy that does not implement bypass, so it keeps on inserting dead blocks in the cache even though it has learned that these blocks are cache-averse. The absence of a bypass policy explains the poor results of Glider compared to MPPP and the two proposed approaches derived from it.

4) *GAP benchmarks*: Based on the characteristics of the reuse distance distribution shown in Table III and Figure 3 for the GAP benchmark suite, these graph processing workloads show a very cache-averse behavior with an average reuse distance of 907.89, which is around 5.8 times higher than the one of *SPEC* workloads, and a proportion of dead block accesses of 55.84 %. Such behavior is expected as GAP workloads are executing graph processing algorithms and are known to traverse vast amount of data in a very unpredictable way. Table III shows a median of 28 accesses, which provides us with useful information as 50 % of the blocks accessed experience a reuse distance of 28 or fewer accesses. This characteristic suggests that having higher LLC associativity could help to achieve higher performance.

In addition, these benchmarks show very high standard deviation on reuse distances, which suggests that the CPU triggers accesses to both cache-averse and cache-friendly blocks with relatively uniform probabilities, which results in the eviction of useful blocks. Thus, we deduce that using a bypass policy like MPPP does would provide performance benefits. Results shown for GAP benchmarks on Figure 2 highlight this property of graph processing workloads.

As we observe that GAP benchmarks with inputs *kron* and *urand* show dramatically low reuse of cache blocks during execution, we categorize GAP benchmarks in two categories: (i) GAP workloads using *kron* and *urand* inputs; (ii) GAP workloads using other inputs.

As figure 3 shows, workloads using *kron* and *urand*

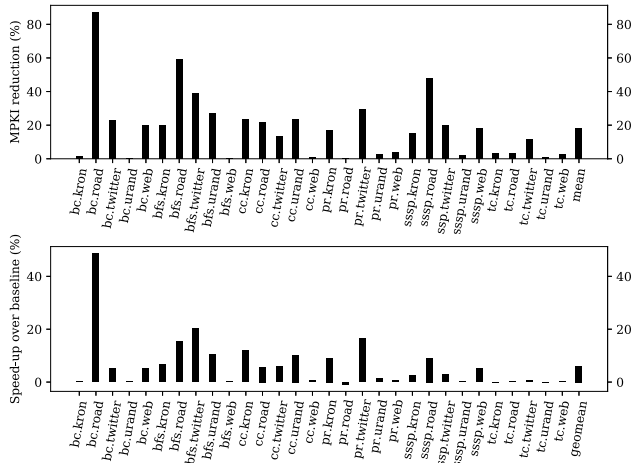


Fig. 4. LLC MPKI reduction and speedup of a 16 MiB LLC over the baseline 2 MiB LLC.

inputs tend to stress more the LLC. As stated by Beamer *et al.* in the specification of the benchmark, these inputs present a very high memory footprint compared to the other inputs used in this work and, moreover, their graphs have a very distinct topology. Urand represents a worst-case as every vertex has an equal probability of being a neighbour of every other vertex. Thus, the graph processing kernel in charge of traversing these inputs has to request memory blocks very distant from another, which eventually causes poor reuse.

As a result, we see that LLC replacement policies are impractical solutions in this context as these workloads are biased towards a cache-averse behavior, all accesses being cache-averse. Although replacement policies cannot improve performance for these workloads, an excellent way to cope with these workloads would be to increase the capacity of the cache substantially or to incorporate a prefetcher that can predict the access patterns to the memory blocks and bring them to the cache before they are accessed.

To demonstrate the need for increased cache capacity, we increase the size of the LLC from 2 MiB to 16 MiB and present the results in Figure 4. Results show a clear benefit from the increased capacity of the LLC. The average MPKI over the whole set of GAP benchmarks drops by 18%, which delivers a geometric mean speedup of 6.1% compared with a 2 MiB LLC with an LRU replacement policy. However, some workloads show reduced MPKI while suffering from an IPC slow-down. The reason for this phenomenon are the DRAM latencies. While increasing the LLC capacity, these workloads have similar statistics in terms of cache accesses and miss rates, but the miss rate in the DRAM row buffers increases. We interpret this behavior as a symptom of workloads with extremely poor temporal and spatial locality.

This clearly shows the need for improved memory allocation policies when it comes to graph processing algorithms. To illustrate this example, we focus on `pr.road`, as Page Rank is an algorithm that should show better locality than the others. When computing the score of an edge of the graph, the

algorithm only visits neighbouring edges to that edge. Thus, the only way for the DRAM to exhibit higher row buffer miss rate is that neighbouring edges in graphs are stored in very distant places in main memory. This showcases the need for a memory allocation policy that takes into account the topology of the graph. Another way to improve the performance of these graph processing workloads would be to incorporate a dedicated on-chip storage structure able to serve, at a low cost, not only the required edges but also their neighbours.

To wrap up this discussion about GAP benchmark suite, the presented results show that there is still a good amount of work to be done on the algorithmic and software side of these workloads to make them cache-friendly. In addition, the reported characteristics of these benchmarks are hard to exploit by traditional cache hierarchies and cache replacement policies, so different solutions should be explored on the hardware side to improve the performance of these benchmarks.

## VIII. CONCLUSIONS

For many years, advances in cache replacement policies have provided important performance improvements. However, many techniques proposed in the literature have considered a reduced amount of CPU-centric benchmarks for their evaluations. Emerging big data and HPC workloads present very different behaviors than traditional benchmarks, specially in terms of memory access patterns. Thus, cache replacement policies need to be re-evaluated to understand their benefits and limitations and to extend the performance benefits they provide to a wider range of workloads.

This paper characterizes different benchmark suites and evaluates state-of-the-art cache replacement policies with these workloads. In particular, we evaluate a mix of graph processing, scientific and industrial workloads (GAP, XSBench and Qualcomm) along with the SPEC CPU 2006 and 2017 benchmark suites, and we observe that the replacement policies provide significant improvements over LRU for the SPEC workloads, but they are not effective in capturing the complex access patterns of HPC and big data workloads.

This paper also proposes two perceptron-based replacement policies that provide good performance across all the considered workloads by dynamically adapting the prediction mechanisms in each program phase.

## ACKNOWLEDGMENTS

This work has been partially supported by the European Union’s Horizon 2020 research and innovation program under the Mont-Blanc 2020 project (grant agreement 779877). Lluc Alvarez has been partially supported by the Spanish Ministry of Economy, Industry and Competitiveness under the Juan de la Cierva Formacion fellowship number FJCI-2016-30984. Marc Casas has been partially supported by the Spanish Ministry of Economy, Industry and Competitiveness under Ramon y Cajal fellowship number RYC-2017-23269. This research was supported by National Science Foundation grant CCF-1912617, Semiconductor Research Corporation project 2936.001, and generous gifts from Intel Labs.

## REFERENCES

- [1] Championship value prediction (CVP). [Online]. Available: <https://www.microarch.org/cvp1/>
- [2] J. Abella, A. González, X. Vera, and M. F. P. O’Boyle, “Iatac: A smart predictor to turn-off l2 cache lines,” *ACM Trans. Archit. Code Optim.*, vol. 2, no. 1, p. 55–77, Mar. 2005. [Online]. Available: <https://doi.org/10.1145/1061267.1061271>
- [3] An-Chow Lai, C. Fide, and B. Falsafi, “Dead-block prediction dead-block correlating prefetchers,” pp. 144–154, June 2001.
- [4] S. Beamer, K. Asanović, and D. Patterson, “The GAP benchmark suite.” [Online]. Available: <http://arxiv.org/abs/1508.03619>
- [5] N. Beckmann and D. Sanchez, “Maximizing cache performance under uncertainty,” in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, pp. 109–120. [Online]. Available: <http://ieeexplore.ieee.org/document/7920818/>
- [6] L. A. Belady, “A study of replacement algorithms for a virtual-storage computer,” pp. 78–101. [Online]. Available: <https://doi.org/10.1147/sj.52.0078>
- [7] ChampSim, “ChampSim/ChampSim,” original-date: 2017-06-30T05:41:50Z. [Online]. Available: <https://github.com/ChampSim/ChampSim>
- [8] N. Duong, D. Zhao, T. Kim, R. Cammarota, M. Valero, and A. V. Veidenbaum, “Improving cache management policies using dynamic reuse distances,” in *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, Dec 2012, pp. 389–400.
- [9] P. V. Gratz, J. Kim, and G. Chacon. THE 2nd CACHE REPLACEMENT CHAMPIONSHIP – co-located with ISCA june 2017. [Online]. Available: <https://crc2.ece.tamu.edu/>
- [10] A. Jain and C. Lin, “Back to the future: Leveraging belady’s algorithm for improved cache replacement,” in *Proceedings of the 43rd International Symposium on Computer Architecture*, ser. ISCA ’16. IEEE, pp. 78–89. [Online]. Available: <https://dl.acm.org/citation.cfm?doid=3007787.3001146>
- [11] A. Jaleel, “Memory characterization of workloads using instrumentation-driven simulation,” *Web Copy: http://www.glue.umd.edu/ajaleel/workload*, 2010.
- [12] A. Jaleel, K. B. Theobald, S. C. S. Jr, and J. Emer, “High performance cache replacement using re-reference interval prediction (RRIP),” in *Proceedings of the 37th annual international symposium on Computer architecture*, vol. ISCA’10. IEEE, pp. 60–71. [Online]. Available: <https://dl.acm.org/citation.cfm?doid=1815961.1815971>
- [13] D. A. Jiménez, “Insertion and promotion for tree-based PseudoLRU last-level caches,” in *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture - MICRO-46*. ACM Press, pp. 284–296. [Online]. Available: <http://dl.acm.org/citation.cfm?doid=2540708.2540733>
- [14] D. A. Jiménez and E. Teran, “Multiperspective reuse prediction,” in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture - MICRO-50 ’17*, ser. MICRO ’17. IEEE, pp. 436–448. [Online]. Available: <http://dl.acm.org/citation.cfm?doid=3123939.3123942>
- [15] S. Khan and D. A. Jimenez, “Insertion policy selection using decision tree analysis,” in *2010 IEEE International Conference on Computer Design*. IEEE, pp. 106–111. [Online]. Available: <http://ieeexplore.ieee.org/document/5647608/>
- [16] S. M. Khan, Y. Tian, and D. A. Jimenez, “Sampling dead block prediction for last-level caches,” in *2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE, pp. 175–186. [Online]. Available: <http://ieeexplore.ieee.org/document/5695535/>
- [17] M. Kharbutli and Y. Solihin, “Counter-based cache replacement and bypassing algorithms,” *IEEE Transactions on Computers*, vol. 57, no. 4, pp. 433–447, April 2008.
- [18] H. Liu, M. Ferdman, J. Huh, and D. Burger, “Cache bursts: A new approach for eliminating dead blocks and increasing cache efficiency,” in *2008 41st IEEE/ACM International Symposium on Microarchitecture*, Nov 2008, pp. 222–233.
- [19] E. Perelman, G. Hamerly, M. V. Biesbrouck, T. Sherwood, and B. Calder, “Using SimPoint for accurate and efficient simulation,” p. 2.
- [20] M. K. Qureshi, A. Jaleel, Y. N. Patt, S. C. Steely, and J. Emer, “Adaptive insertion policies for high performance caching,” in *Proceedings of the 34th Annual International Symposium on Computer Architecture*, ser. ISCA ’07. IEEE, pp. 381–391. [Online]. Available: <http://doi.acm.org/10.1145/1250662.1250709>
- [21] Z. Shi, X. Huang, A. Jain, and C. Lin, “Applying deep learning to the cache replacement problem,” in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO ’52. New York, NY, USA: Association for Computing Machinery, 2019, p. 413–425. [Online]. Available: <https://doi.org/10.1145/3352460.3358319>
- [22] D. Tarjan, K. Skadron, and M. Stan, “An ahead pipelined alloyed perceptron with single cycle access time,” in *Proceedings of the 20th Annual International Symposium on Computer Architecture*, p. 8.
- [23] E. Teran, Y. Tian, Z. Wang, and D. A. Jimenez, “Minimal disturbance placement and promotion,” in *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, pp. 201–211. [Online]. Available: <http://ieeexplore.ieee.org/document/7446065/>
- [24] E. Teran, Z. Wang, and D. A. Jimenez, “Perceptron learning for reuse prediction,” in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, ser. MICRO ’16. IEEE, pp. 1–12. [Online]. Available: <http://ieeexplore.ieee.org/document/7783705/>
- [25] J. R. Tramm, A. R. Siegel, T. Islam, and M. Schulz, “XSBench – the development and verification of a performance abstraction for monte carlo reactor analysis,” in *PHYSOR 2014 - The Role of Reactor Physics toward a Sustainable Future*, p. 12. [Online]. Available: <https://www.mcs.anl.gov/papers/P5064-0114.pdf>
- [26] B. Widrow and M. Lehr, “30 years of adaptive neural networks: perceptron, madaline, and backpropagation,” vol. 78, no. 9, pp. 1415–1442. [Online]. Available: <http://ieeexplore.ieee.org/document/58323/>
- [27] C.-J. Wu, A. Jaleel, W. Hasenplaugh, M. Martonosi, S. C. Steely, and J. Emer, “SHiP: signature-based hit predictor for high performance caching,” in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture - MICRO-44 ’11*. ACM Press, p. 430. [Online]. Available: <http://dl.acm.org/citation.cfm?doid=2155620.2155671>
- [28] W. A. Wulf and S. A. McKee, “Hitting the memory wall: implications of the obvious,” vol. 23, no. 1, pp. 20–24. [Online]. Available: <http://portal.acm.org/citation.cfm?doid=216585.216588>
- [29] X. Yu, C. J. Hughes, N. Satish, and S. Devadas, “Imp: Indirect memory prefetcher,” in *Proceedings of the 48th International Symposium on Microarchitecture*, ser. MICRO-48. New York, NY, USA: Association for Computing Machinery, 2015, p. 178–190. [Online]. Available: <https://doi.org/10.1145/2830772.2830807>
- [30] Y. Zhang, V. Kiriansky, C. Mendis, S. Amarasinghe, and M. Zaharia, “Making caches work for graph analytics,” in *2017 IEEE International Conference on Big Data (Big Data)*, Dec 2017, pp. 293–302.
- [31] Zhigang Hu, S. Kaxiras, and M. Martonosi, “Timekeeping in the memory system: predicting and optimizing memory behavior,” in *Proceedings 29th Annual International Symposium on Computer Architecture*, May 2002, pp. 209–220.